Week 9: Bisection method, Newton's Method Gradient Descent

Ziwen Ye

Stevens Institute of Technology

zye2@stevens.edu

November 3, 2018

Outline

Newton's Method

- The Newton's Method, is a powerful technique for solving equations numerically.
- It is based on the Taylor Series.
- Usually converge on a root with devastating efficiency.

Newton's Method

Assume r is one root of function $f(x)$. Let x_0 be a "good guess" of r, $r = x_0 + h$. Then we have the following:

Newton's Method

$$f(r) = f(x_0 + h) = f(x_0) + h * f'(x_0) \tag{1}$$

$$h = -\frac{f(x_0)}{f'(x_0)} \tag{2}$$

$$r = x_0 + h \approx x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3}$$

Newton's Method

Actually, by Taylor's expansion:

$$
\begin{aligned}
f(x) &= \sum_{i=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \\
&= f(a) + f'(a)(x-a) + \ldots
\end{aligned}
$$

Substitute $x = r, a = x_0$, we have:

$$f(r) \approx f(x_0) + h * f'(x_0) \tag{4}$$

Newton's Method

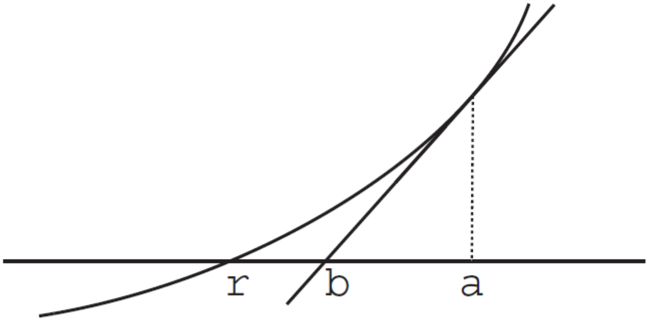
Our new improved estimate x_1 is therefore given by

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Continue in this way, if x_n is the current estimate, then the next estimate x_{n+1} is given by:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Geometrical Interpretation



Newton's Method

Here is a simple example: $f(x) = x^2 - 2$

$$x_{n=1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - 2}{2x_n}$$

- $x_0 = 1,\ \Delta x = \frac{1-2}{2} = -0.5,\ x_1 = 1.5$
- $x_1 = 1.5,\ \Delta x = \frac{1.5^2-2}{3} = 0.08,\ x_2 = 1.42$
- $x_2 = 1.42,\ \Delta x = \frac{1.42^2-2}{2.84} = 0.006,\ x_3 = 1.414$

Theoretically,

$$x^2 - 2 = 0, x = \pm\sqrt{2} \approx \pm 1.414$$

Newton's Method

Algorithm

```
loop {
```

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
    # until converge

}
```

Newton's Method

Compare with gradient, we have almost the same implementation in R:

Example (pseudo code)

```
# set initial
# set convergence condition
# loop:
while(1)
{
    # algorithm (probably the only difference)
    # ...
    # if (converge)
    # {
    #    break
    # }
}
```

Newton's Method in R

Use Newton's Method to solve the root of a function: $f(x) = x^2 - 2$

Example

```
# f(x) = x^2 - 2
fx <- function(x)
{
    y = x^2 - 2
    return (y)
}

# f'(x) = 2*x
dfx <- function(x)
{
    y = 2*x
    return(y)
}
```

Newton's Method in R

Example

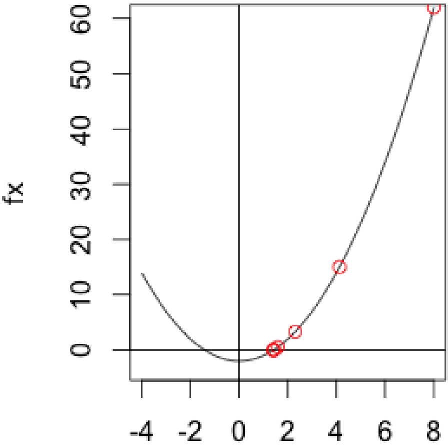
```
# initial plot
plot(fx, ylim = c(-3, 60), xlim = c(-4, 8))
abline(h=0, v=0)

x0 = 8              # initial guess
epsilon = 0.01      # convergent condition
points(x0, fx(x0), col = "red")

step = 1            # step count
```

Newton's Method in R

Example

```
while(1)
{
    tmp = x0
    deltaX = fx(x0) / dfx(x0)
    x0 = x0 - deltaX
    points(x0, fx(x0), col = "red")
    step = step + 1
    # convergence
    if (abs(x0-tmp) < epsilon)
    {
        points(x0, fx(x0), col = "red")
        print(paste("x0 = ", x0, ", step = ", step, sep=""))
        break
    }
}
```

Newton's Method in R



Example: Calculate the Bond Yield

We have a bond, paying coupon of \$3 every 6 months. The maturity is 2 years and the face value is \$100. If the bond price is \$98.39, calculate the yield of the bond using Newton's Method.

$$3e^{-0.5y} + 3e^{-1y} + 3e^{-1.5y} + 103e^{-2y} = 98.39$$

- $f(y) = 3e^{-0.5y} + 3e^{-1y} + 3e^{-1.5y} + 103e^{-2y} - 98.39$
- $\frac{df}{dy} = -1.5e^{-0.5y} - 3e^{-1y} - 4.5e^{-1.5y} - 206e^{-2y}$

Example: Calculate the Bond Yield

Example

```
bond <- function(y) {
value <- 3 * exp(-0.5 * y) + 3 * exp(-1 * y) +
         3 * exp(-1.5 * y) + 103 * exp(-2 * y) - 98.39
return(value)
}

dbond <- function(y) {
value <- -1.5 * exp(-0.5 * y) + -3 * exp(-1 * y) +
 -4.5 * exp(-1.5 * y) + -206 * exp(-2 * y)
return(value)
}
```

Example: Calculate the Bond Yield

Example

```
y0 = 0.02
while(1)
{
    y1 <- y0 - bond(y0) / dbond(y0)
    if(abs(y0-y1) < 1e-5)
    {
        print("converged!")
        cat("y = ", y1, sep = "")
        break
    }
    y0 <- y1
}
```

Gradient

Definition

If we have a bivariate function $f(x, y)$, then the partial derivatives, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ are the rate of change of f with respect to x and y.

We put them together in a vector, and call it *Gradient of f*:

$$\nabla f = \langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \rangle$$

Gradient

- Of course, if we specify a point P_0 and we can calculate the gradient on that point:

$$\nabla f|_{P_0} = \langle \frac{\partial f}{\partial x}|_{P_0}, \frac{\partial f}{\partial y}|_{P_0} \rangle$$

- For functions with only one variable, the gradient equals to the derivative.
- Similarly, for higher dimensional functions, for example $f(x_1, ..., x_n)$, we have:

$$\nabla f = \langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \rangle$$

Gradient Descent Algorithm

Algorithm: Single Variate Function

```
loop {
```

$$x := x - \alpha \frac{df}{dx}$$

```
    # until converge
}
```

Algorithm: Multi Variate Function

```
loop {
```

$$\mathbf{x} := \mathbf{x} - \alpha \nabla f$$

```
    # until converge
}
```

where α is named "step size" or "learning rate".

Gradient Descent Algorithm

Analysis:

- if $df/dx > 0$, which means $x_0 > x_{min}$, $x - \alpha\frac{df}{dx} \downarrow$
- if $df/dx < 0$, which means $x_0 < x_{min}$, $x - \alpha\frac{df}{dx} \uparrow$
- x_0 will converge to the extrema in no matter which case.
- The step size α also determines the speed of convergence.

Gradient Descent in R

Let's see how to implement this algorithm in R.

Example (pseudo code)

```
# set initial
# set convergence condition
# loop:
while(1)
{
    # algorithm
    # ...
    # if (converge)
    # {
    #    break
    # }
}
```

Gradient Descent in R

First define the function to be optimized: $f(x) = x^2 - 10x + 3$

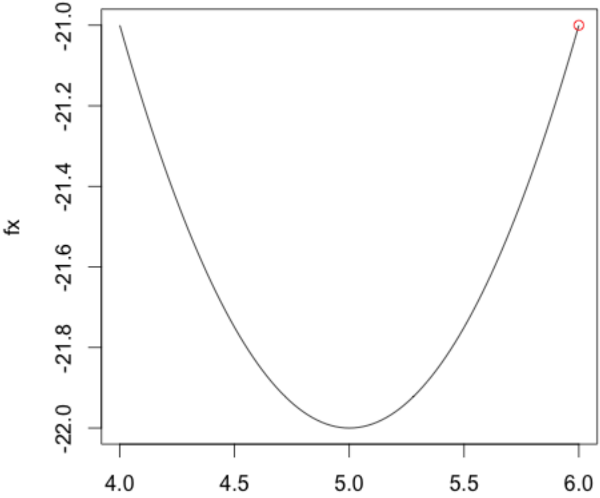
Example

```
> # function f(x) = x^2 - 10x + 3
> # x.min = -b/2a = 5
> fx <- function(x) {
+     y = x^2 - 10 * x + 3
+     return (y)
+ }
> # derivative of f(x)
> # df = x*2 - 10
> df <- function(x) {
+     y = x*2 - 10
+     return (y)
+ }
```

Example

```
> plot(fx, xlim = c(4, 6))

> x0 = 6        # initial value
> points(x0, fx(x0), col = "red")

> alpha = 0.2  # step length

> epsilon = 0.0001 # condition to terminate the algorithm
> # step count
> step = 1
```

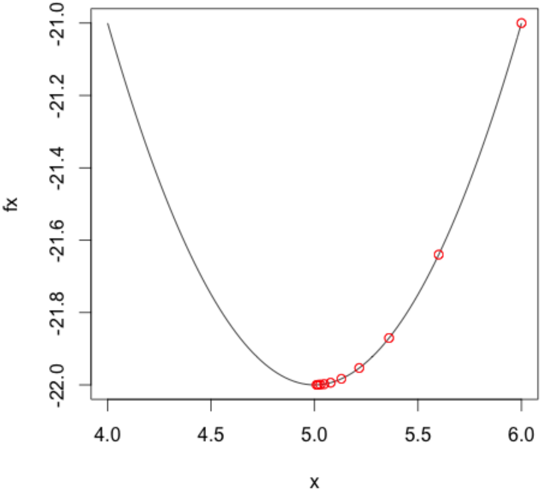
Gradient Descent in R



Example

```
> while(1)
+ {
+     cat("Calculating, step ", step, '\n', sep = "")
+     x1 = x0 - alpha * df(x0)    # update x
+
+     # check convergence
+     if (abs(fx(x1) - fx(x0)) < epsilon)
+     {
+         cat("x = ", x1, '\n', sep='')
+         cat("Final step: ", step, '\n', sep='')
+         break
+     }
+     points(x1, fx(x1), col = "red")
+     x0 = x1
+     step = step + 1
+     Sys.sleep(1.2) # suspend for a while
+ }
```

Gradient Descent in R



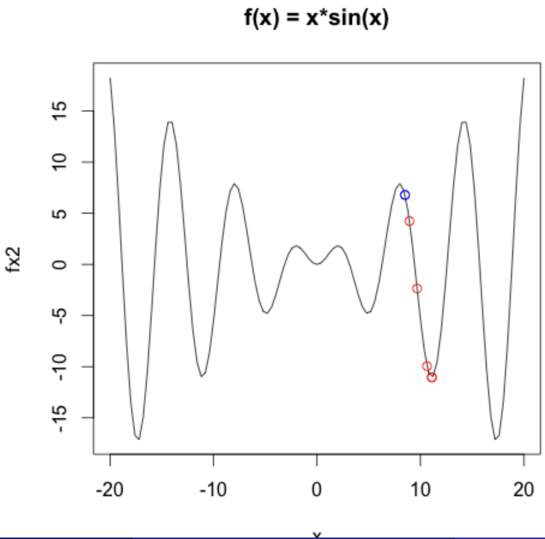
Gradient Descent in R

Two potential issue with this algorithm

- The size of α
- Local extrema.

Local Extrema

Local extrema, see details in the code



Black-Scholes-Merton model

Black-Scholes-Merton model is a mathematical model for the dynamics of a financial market containing derivative investment instruments.

In most cases, this model is used to calculate the option price under risk-free measure. Five variables are needed to calculate the option price

- S_t underlying asset price
- K option strike price
- τ time to maturity
- r risk-free rate
- σ volatility

It can also be used to calculate implied volatility in the reversed way.

The model

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r\tau}$$
$$P(S_t, t) = N(-d_2)Ke^{-r\tau} - N(-d_1)S_t$$
$$d_1 = \frac{1}{\sigma_n\sqrt{\tau}}[ln(\frac{S_t}{K}) + (r + \frac{\sigma_n^2}{2})\tau]$$
$$d_2 = d_1 - \sigma_n\sqrt{\tau}$$